Vincentius Charles Maynad
07111940000038
SPC

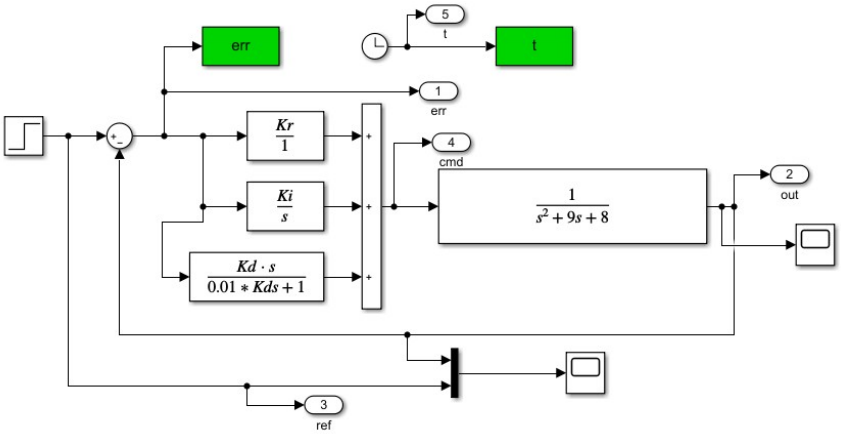
TUGAS 6 SISTEM PENGATURAN CERDAS

GENETIC ALGORITHM

Pengantar dan Pengenalan Sistem

Genetic Algorithm atau biasa disingkat dengan GA adalah algoritma optimasi global stokastik. Algoritma ini adalah jenis algoritma evolusioner yang terinspirasi dari proses seleksi alam dengan representasi biner dan operator sederhana berdasarkan rekombinasi serta mutasi genetik. Beberapa pemanfaatan GA dalam sistem kontrol adalah kontrol *multi – object*, kontrol PID, kontrol optimasi, dan sebagainya.

Pada tugas ini, saya akan mencoba mengimplementasikan GA untuk menentukan parameter dari kontroler PID untuk plant dari tugas sebelumnya. Sistem plant – nya dapat dilihat pada gambar.



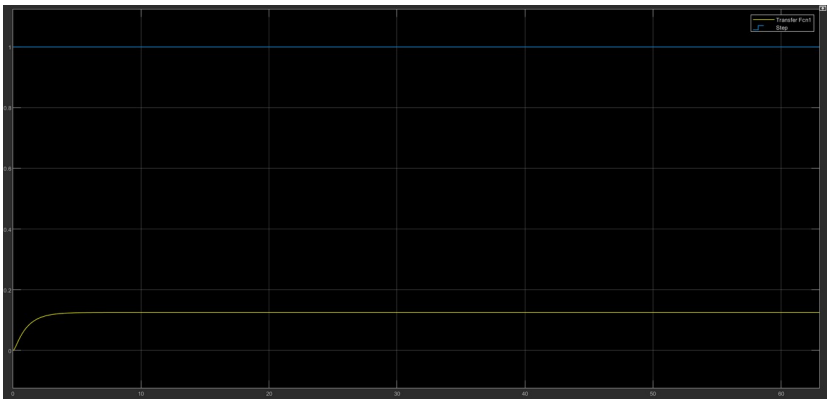
Gambar 1. Model plant motor

Model plant yang dipakai adalah

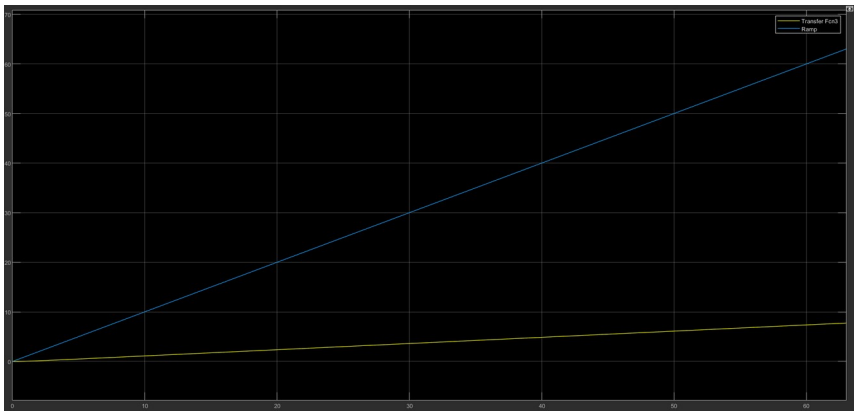
$$G(s)=\frac{1}{s^2+9s+8}$$

Pengujian Plant

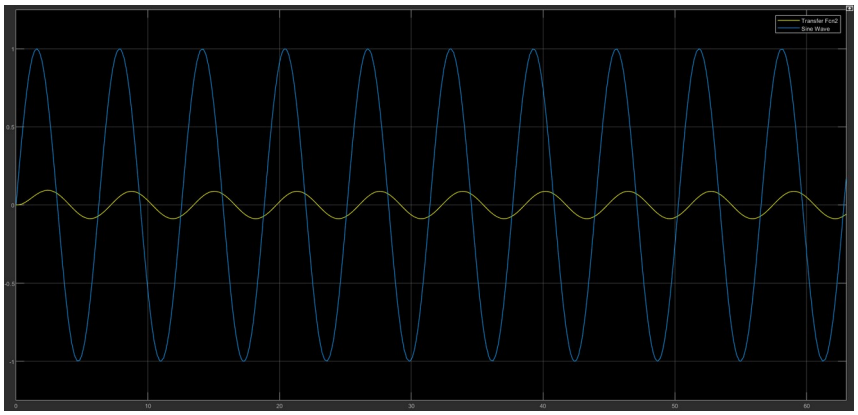
Untuk melihat respon dari plant, kita coba lakukan pengujian dengan input step, ramp, dan sinusoidal.



Gambar 2. Respon step *open loop sistem*.



Gambar 3. Respon ramp *open loop sistem*.



Gambar 4. Respon sine *open loop sistem*.

Dari ketiga respon, dapat dilihat kalau output dari plant belum sesuai dengan model inputnya. Pada uji step, terlihat kalau output yang didapat tidak memiliki *overshot*, namun bernilai *steady state* tidak sama dengan 1. Hal itu didukung dengan gambar 3, setelah beberapa detik, nilai output dibawah nilai yang seharusnya. Sedangkan dari uji sinusoidal, hanya terlihat perbedaan fasa sedikit. Oleh karena itu, kita membutuhkan kontroler untuk menyesuaikan hasil output sesuai yang kita inginkan.

Metode Kerja

Proses pengolahan GA dilakukan dengan program MATLAB. Berikut prosedur yang perlu dilakukan :

1. **Pemilihan kriteria kontroler**

   Pertama, kita tentukan terlebih dahulu spesifikasi dari kontroler yang kita inginkan, antara lain waktu *settling time* dan batas *overshot* yang diperbolehkan. Kali ini, dipakai :

   - Waktu settling maksimum : 4
   - Waktu settling minimum : 2
   - Batas overshot : 0.001

   Kita tuliskan pada program :

```
maxSettlingTimeCond = 4;        % waktu maksimum settling
minSettlingTimeCond = 2;        % waktu minimum settling
      OvershootCond = 0.001;    % overshoot maksimum;
```

2. **Inisiasi populasi GA**

   Kemudian, kita tentukan terlebih dahulu nilai populasi dari GA. Selain itu, untuk kemudahan dalam pemprogramman, ditentukan pula beberapa parameter lain.

   - Konstrain GA = 10
   - Ukuran populasi = 80
   - Batas dari gain kontroler
     - Kp = 0.01 sampai 20
     - Ki = 0.01 sampai 20
     - Kd = 0.01 sampai 20
   - Epsilon = 10^{-6}

   Kita tuliskan pada program :

```
evalFN = fitnessFunctionName;
termFNOptimalValue = 10; % konstraint untuk pemutusan perhitungan GA
populationSize = 80;
initBounds = [0.01,20;0.01,20;0.01,20];
epsilon = 1e-6;
options = [epsilon 1];
startPop = initializega(populationSize,initBounds,evalFN,evalOps,options);
```

   evalFN adalah fungsi MATLAB untuk mendeklarasikan fungsi objektif. Initbounds adalah batas bawah dan atas gain. Fungsi initializega adalah fungsi untuk menghasilkan populasi awal dengan 80 anggota. Nilainya dibatasai oleh initBounds yang berpresisi 6 desimal sesuai varibel options.

3. **Pemilihan parameter GA**

   Kita juga inisiasi beberapa parameter lain yang ada di GA pada program, yaitu..

```
generalBounds = [0.01,20;0.01,20;0.01,20];
termFN = 'optMaxGenTerm';
selectFN = 'normGeomSelect';
selectOps = 0.08;
xOverFNs = 'arithXover';
xOverOps = 4;
mutFNs = 'unifMutation';
mutOps = 8;
```

   General bounds menunjukan ruang pencarian dari GA, termFN mengarah ke fungsi terminasi dari GA yang akan berhenti bila nilai parameter yang kita inginkan sudah terpenuhi. SelectFN adalah fungsi untuk proses seleksi dan selectOps adalah probabilitas pemilihan kromosom terkuat (fittest) pada tiap generasi. xOverFNs adalah pemilihan fungsi aritmatika sebagai prosedur crossover. Sedangkan xOverOps adalah spesifikasi jumlah titik crossover. mutFNs adalah fungsi mutasi.

4. **Perhitungan oleh GA**

Proses dari GA akan terus terjadi secara rekursif, sampai nilai batas parameter yang kita inginkan terpenuhi. Kita gunakan sebuah fungsi untuk melakukan GA, yaitu..

```
[bestIndividual,endPop,bPop,traceInfo] = ga(varBounds,evalFN,evalOps, ...
    startPop,opts,termFN,termOps,selectFN,selectOps, ...
    xOverFNs,xOverOps,mutFNs,mutOps);
```

Setelah proses selesai, output yang dihasilkan adalah

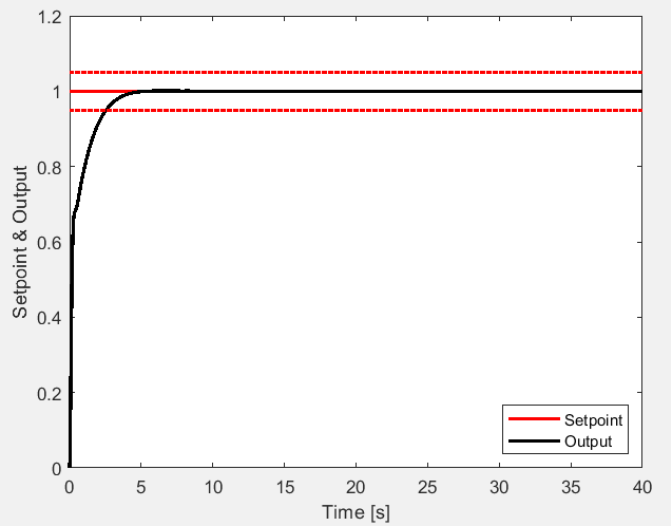
- bestindividual : populasi terbaik yang didapatkan dari proses.
- endPop : final populasi dari GA
- bPop : solusi terbaik dari GA dari seluruh generasi
- traceInfo : nilai terbaik dan rerata nilai tiap generasi

Hasil Simulasi

Berdasarkan proses yang sudah dilakukan, kita dapatkan hasil sebagai berikut..

- K_p=19.9635
- K_i=13.7219
- K_d=8.1727

Didapatkan perbandingan output dan input



Gambar 5. Perbandingan input dan output dengan 80 populasi

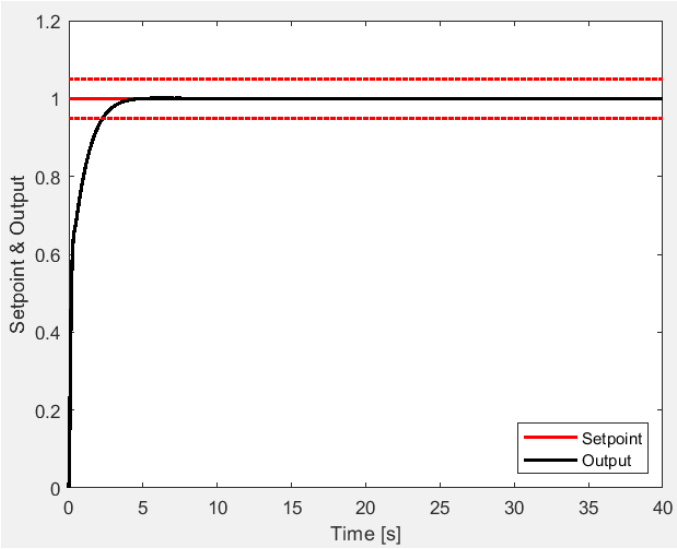
Grafik tersebut memiliki *settling time* 2.49 sekon dan overshot pas di 0.1%. Adapun *steady state error* – nya adalah 0. Hasil ini didapatkan dengan 80 populasi yang dihitung selama 100 generasi.

Sekarang kita coba lagi perhitungan 100 generasi namun dengan jumlah populasi 100. Didapatkan hasil..

- K_p=19.9205

- K_i=14.47
- K_d=6.9246

Dengan *settling time* 2,08 sekon dan overshot juga di 0.1% serta *steady state error* – nya juga 0.



Gambar 5. Perbandingan input dan output dengan 100 populasi

Kesimpulan

Dari simulasi yang dilakukan bisa disimpulkan kalau pemakaian GA dengan jumlah populasi 100 lebih baik daripada populasi yang berjumlah 80 bila dilihat dari lama *settling time* – nya.

Vincentius Charles Maynad
07111940000038
SPC

 # PROGRAM #

##

```
function [bestIndividual, endPop, bPop, NumberOfGenerations] = ...
    GA_ControllerDesign(criteria, fitnessFunctionName, ...
    mdlFileToUse, matFileToUse)
global indexType
global mdlFile
global simulationTime
global maxSettlingTimeCond
global minSettlingTimeCond
global OvershootCond


load(matFileToUse);
indexType = criteria;
mdlFile = mdlFileToUse;

simulationTime = 40;
warning('off');
%_____________________________________________________________________
% parameter awal GA
evalFN = fitnessFunctionName;
termFNOptimalValue = 10;
populationSize = 80;

[bestIndividual,endPop,bPop,traceInfo]=GAOT_ECM_ControllerDesign_GA( ...
    criteria, evalFN, termFNOptimalValue, initBounds, generalBounds, ...
    populationSize);
%_____________________________________________________________________
% Analisis output hasil perhitungan GA
NumberOfGenerations = traceInfo(length(traceInfo(:,1)),1);
fprintf('Number of Generations = %d\n', NumberOfGenerations);
Kr=bestIndividual(1);
Ki=bestIndividual(2);
fprintf('\nCotroller Parameters:');
fprintf('\n  Kr = %.2f \n  Ki = %.2f', Kr, Ki);

assignin('base', 'Kr', bestIndividual(1));
assignin('base', 'Ki', bestIndividual(2));
[T,state,outputs]=sim(mdlFile,simulationTime);
err = outputs(:,1);
t = outputs(:,5);
yt1=find(err>0.05, 1, 'last');
yt2=find(err<-0.05, 1, 'last');
SettlingTime = t(max([yt1 yt2]));
Overshoot = abs(min(min(err),0));
SSErr = mean(err(max([yt1 yt2])+1:end));     % error steady state
fprintf('\n\nPerformance of Controller:');
fprintf('\n  Settling Time: %.2f [s]', SettlingTime);
fprintf('\n  Overshoot: %.1f [%%]', Overshoot*100);
```

```
fprintf('\n  Steady State Error: %.0f', SSErr);
fprintf('\n\nRequired Performance:');
fprintf('\n  Maximum Settling Time: %.2f [s]',maxSettlingTimeCond);
fprintf('\n  Minimum Settling Time: %.2f [s]', minSettlingTimeCond);
fprintf('\n  Maximum Overshoot: %.1f [%%]',OvershootCond*100);
fprintf('\n  Steady State Error: 0\n');
figure
plot(T, outputs(:,3), 'r', T, outputs(:,2), 'k', ...
    T, ones(length(T),1)*1.05, ':r', T, ones(length(T),1)*0.95, ':r', ...
    'LineWidth',2);
axis([0 simulationTime 0 1.2]);
xlabel('Time [s]');
ylabel('Setpoint & Output');
legend('Setpoint','Output','Location','SouthEast');
end

##

function [bestIndividual, endPop, bPop, traceInfo] = ...
    ControllerDesign(criteria, evalFN, ...
    termFNOptimalValue, initBounds, varBounds, populationSize)
global indexType
indexType = criteria;
warning('off');
%_____________________________________________________________________
% STEP 1: Inisialisasi GA
NoGenerations = 100;
evalOps = [];
epsilon = 1e-6;
options = [epsilon 1];
displayOpt = 0;
% inisialisasi populasi awal
startPop = initializega(populationSize,initBounds,evalFN,evalOps,options);
if (displayOpt == 1)
    display('Initialization complete.');
end
%_____________________________________________________________________
% STEP 2: definisikan paramter GA
prob_ops = 1;
opts = [epsilon prob_ops displayOpt];
termOps = [NoGenerations termFNOptimalValue epsilon];
termFN = 'optMaxGenTerm';
selectFN = 'normGeomSelect';
selectOps = 0.08;
xOverFNs = 'arithXover';
xOverOps = 4;
mutFNs = 'unifMutation';
mutOps = 8;
%_____________________________________________________________________
% STEP 3: Iterasi perhitungan GA.
if (displayOpt == 1)
    display('Starting GA loop...');
end
[bestIndividual,endPop,bPop,traceInfo] = ga(varBounds,evalFN,evalOps, ...
    startPop,opts,termFN,termOps,selectFN,selectOps, ...
    xOverFNs,xOverOps,mutFNs,mutOps);
if (displayOpt == 1)
    display('Finished!');
```

```
end
%_____________________________________________________________________
end

##

function [pop] = initializega(num, bounds, evalFN,evalOps,options)


if options(2)==1 %Float GA
  estr=['[ pop(i,:) pop(i,xZomeLength)]=' evalFN '(pop(i,:),[0
evalOps]);'];
else %Binary GA
  estr=['x=b2f(pop(i,:),bounds,bits);[x v]=' evalFN ...
    '(x,[0 evalOps]); pop(i,:)=[f2b(x,bounds,bits) v];'];
end


numVars     = size(bounds,1);       %jumlah variabel ( Kp,Ki,Kd)
rng         = (bounds(:,2)-bounds(:,1))'; %batasan variabel

if options(2)==1 %Float GA
  xZomeLength = numVars+1;
  pop         = zeros(num,xZomeLength);     %Alokasi populasi baru
  pop(:,1:numVars)=(ones(num,1)*rng).*(rand(num,numVars))+...
    (ones(num,1)*bounds(:,1)');
else %Binary GA
  bits=calcbits(bounds,options(1));
  xZomeLength = sum(bits)+1;
  pop = round(rand(num,sum(bits)+1));
end

for i=1:num
  eval(estr);
end

##

function [x,endPop,bPop,traceInfo] =
ga(bounds,evalFN,evalOps,startPop,opts,...
termFN,termOps,selectFN,selectOps,xOverFNs,xOverOps,mutFNs,mutOps)

n=nargin;
if isempty(opts)
  opts = [1e-6 1 0];
end

if opts(2)==1 %Float ga
  e1str=['[c1 c1(xZomeLength)]=' evalFN '(c1,[gen evalOps]);'];
  e2str=['[c2 c2(xZomeLength)]=' evalFN '(c2,[gen evalOps]);'];
else %Binary ga
  e1str=['x=b2f(endPop(j,:),bounds,bits);[x v]=' evalFN ...
    '(x,[gen evalOps]); endPop(j,:)=[f2b(x,bounds,bits) v];'];
end

if isempty(startPop) % jika populasi tidak dideklarasikan, buat baru
  startPop=initializega(80,bounds,evalFN,evalOps,opts(1:2));
```

```
end

if opts(2)==0 %binary
  bits=calcbits(bounds,opts(1));
end

xOverFNs=parse(xOverFNs);
mutFNs=parse(mutFNs);

xZomeLength  = size(startPop,2);    %panjang chromosome
numVar       = xZomeLength-1;       %jumlah variabel
popSize      = size(startPop,1);    %jumlah individual pada populasi
endPop       = zeros(popSize,xZomeLength); %Populasi ke dua
c1           = zeros(1,xZomeLength);    %parent 1 ( individual )
c2           = zeros(1,xZomeLength);    %parent 2 ( individual )
numXOvers    = size(xOverFNs,1);    % jumlah operator crossover
numMuts      = size(mutFNs,1);      % jumlah operator mutasi
epsilon      = opts(1);                 %Threshold untuk membedakan 2
fitness
oval         = max(startPop(:,xZomeLength)); %nilai terbaik pada populasi
awal
bFoundIn     = 1;           % jumlah perubahan nilai terbaik
done         = 0;
gen          = 1;
collectTrace = (nargout>3);
floatGA      = opts(2)==1;
display      = opts(3);

while(~done)
  %Elitist Model
  [bval,bindx] = max(startPop(:,xZomeLength)); %hasil terbaik pada populasi
sebelumnya
  best =  startPop(bindx,:);

  if collectTrace
    traceInfo(gen,1)=gen;                 %populasi sebelumnya
    traceInfo(gen,2)=startPop(bindx,xZomeLength);       %fitness terbaik
    traceInfo(gen,3)=mean(startPop(:,xZomeLength));     %fitness rata-rata
    traceInfo(gen,4)=std(startPop(:,xZomeLength));
  end

  if ( (abs(bval - oval)>epsilon) | (gen==1)) %jika menemukan solusi
terbaik yang lain
    if display
      fprintf(1,'\n%d %f\n',gen,bval);
    end
    if floatGA
      bPop(bFoundIn,:)=[gen startPop(bindx,:)]; %Update bPop Matrix
    else
      bPop(bFoundIn,:)=[gen b2f(startPop(bindx,1:numVar),bounds,bits)...
      startPop(bindx,xZomeLength)];
    end
    bFoundIn=bFoundIn+1;                      %Update jumlah perubahan
    oval=bval;                                %Update nilai terbaik
  else
    if display
      fprintf(1,'%d ',gen);               %jika tidak menemukan, hanya
update jumlah geerasi
```

```
    end
  end

  endPop = feval(selectFN,startPop,[gen selectOps]); % Proses seleksi

  if floatGA %Running with the model where the parameters are numbers of
ops
    for i=1:numXOvers,
      for j=1:xOverOps(i,1),
    a = round(rand*(popSize-1)+1);  %pilih parent 1
    b = round(rand*(popSize-1)+1);  %Pilih parent 2
    xN=deblank(xOverFNs(i,:));
    [c1 c2] = feval(xN,endPop(a,:),endPop(b,:),bounds,[gen xOverOps(i,:)]);
% Proses mutasi
% Cek hassil mutasi ( memastikan jika yang terbaru berbeda seperti
sebelumnya )
    if c1(1:numVar)==endPop(a,(1:numVar))
      c1(xZomeLength)=endPop(a,xZomeLength);
    elseif c1(1:numVar)==endPop(b,(1:numVar))
      c1(xZomeLength)=endPop(b,xZomeLength);
    else
      eval(e1str);
    end
    if c2(1:numVar)==endPop(a,(1:numVar))
      c2(xZomeLength)=endPop(a,xZomeLength);
    elseif c2(1:numVar)==endPop(b,(1:numVar))
      c2(xZomeLength)=endPop(b,xZomeLength);
    else
      eval(e2str);
    end
    endPop(a,:)=c1;
    endPop(b,:)=c2;
      end
    end

    for i=1:numMuts,
      for j=1:mutOps(i,1),
    a = round(rand*(popSize-1)+1);
    c1 = feval(deblank(mutFNs(i,:)),endPop(a,:),bounds,[gen mutOps(i,:)]);
    if c1(1:numVar)==endPop(a,(1:numVar))
      c1(xZomeLength)=endPop(a,xZomeLength);
    else
      eval(e1str);
    end
    endPop(a,:)=c1;
      end
    end
  else %We are running a probabilistic model of genetic operators
    for i=1:numXOvers,
      xN=deblank(xOverFNs(i,:));
      cp=find(rand(popSize,1)<xOverOps(i,1)==1);
      if rem(size(cp,1),2) cp=cp(1:(size(cp,1)-1)); end
      cp=reshape(cp,size(cp,1)/2,2);
      for j=1:size(cp,1)
    a=cp(j,1); b=cp(j,2);
    [endPop(a,:) endPop(b,:)] = feval(xN,endPop(a,:),endPop(b,:),...
      bounds,[gen xOverOps(i,:)]);
      end
```

```
    end
    for i=1:numMuts
      mN=deblank(mutFNs(i,:));
      for j=1:popSize
    endPop(j,:) = feval(mN,endPop(j,:),bounds,[gen mutOps(i,:)]);
    eval(e1str);
      end
    end
  end

  gen=gen+1;
  done=feval(termFN,[gen termOps],bPop,endPop); %Pengecekan perhitungan GA
  startPop=endPop;          %Swap populasi

  [bval,bindx] = min(startPop(:,xZomeLength)); %penyimpanan solusi terbaik
  startPop(bindx,:) = best;
end

[bval,bindx] = max(startPop(:,xZomeLength));
if display
  fprintf(1,'\n%d %f\n',gen,bval);
end

x=startPop(bindx,:);
if opts(2)==0 %binary
  x=b2f(x,bounds,bits);
  bPop(bFoundIn,:)=[gen b2f(startPop(bindx,1:numVar),bounds,bits)...
      startPop(bindx,xZomeLength)];
else
  bPop(bFoundIn,:)=[gen startPop(bindx,:)];
end

if collectTrace
  traceInfo(gen,1)=gen;
  traceInfo(gen,2)=startPop(bindx,xZomeLength); % fitness terbaik
  traceInfo(gen,3)=mean(startPop(:,xZomeLength)); % fitness rata-rata
end

##

function [done] = optMaxGenTerm(ops,bPop,endPop)
currentGen = ops(1);
maxGen     = ops(2);
optimal    = ops(3);
epsilon    = ops(4);
fitIndex   = size(endPop,2);
bestSolVal = max(endPop(:,fitIndex));
done       = (currentGen >= maxGen) | ((optimal - bestSolVal) <= epsilon);


##

function[newPop] = normGeomSelect(oldPop,options)

q=options(2);               % Probabailitas untuk pemilihan terbaik
e = size(oldPop,2);             % Length of xZome, i.e. numvars+fit
n = size(oldPop,1);             % Jumlah individual
```

Vincentius Charles Maynad
07111940000038
SPC

```
newPop = zeros(n,e);
fit = zeros(n,1);
x=zeros(n,2);
x(:,1) =[n:-1:1]';
[y x(:,2)] = sort(oldPop(:,e));
r = q/(1-(1-q)^n);          % Normalisasi distribusi
fit(x(:,2))=r*(1-q).^(x(:,1)-1);    % Generate probabilitas dari seleksi
fit = cumsum(fit);          % Perhitungan kumulatif fungsi probabilistik
rNums=sort(rand(n,1));          % generate angka random
fitIn=1; newIn=1;
while newIn<=n              % proses mendapatkan individual baru
  if(rNums(newIn)<fit(fitIn))
    newPop(newIn,:) = oldPop(fitIn,:);  % Pemilihan individual
    newIn = newIn+1;            % Mencari indiviudal baru
  else
    fitIn = fitIn + 1;          % Mencari seleksi potensial terbaru
  end
end

##

function [c1,c2] = arithXover(p1,p2,bounds,Ops)
% Pilih koef mix random
a = rand;
% Pembuatan anak
c1 = p1*a     + p2*(1-a);
c2 = p1*(1-a) + p2*a;

##

function [parent] = uniformMutate(parent,bounds,Ops)
df = bounds(:,2) - bounds(:,1);
numVar = size(parent,2)-1;
% Pilih variabel random untuk dimutasi
mPoint = round(rand * (numVar-1)) + 1;
newValue = bounds(mPoint,1)+rand * df(mPoint); % Proses mutasi
parent(mPoint) = newValue;      % Pembuatan anak

##
function [bits]=calcbits(bounds,precision)
bits=ceil(log2((bounds(:,2)-bounds(:,1))' ./ precision));
```